ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 3
Том 88, 1994

ANNUAIRE DE L'UNIVERSITE DE SOFIA „ST. KLIMENT OHRIDSKI“
FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 3
Tome 88, 1994

# АЛГОРИТЪМ ЗА ИЗВОД НА ТИП И РЕАЛИЗАЦИЯТА МУ НА ЕЗИКА W

МАГДАЛИНА ТОДОРОВА, ДИМИТЪР БИРОВ

*Магдалина Тодорова, Димитр Биров.* АЛГОРИТМ ВЫВОДА ТИПА И ЕГО РЕАЛИЗАЦИЯ НА ЯЗЫКЕ *W*

Понятие о полиморфизме в языках программирования характеризирует данные, которые имеют более одного типа, или функции, оперирующие на множество типов. В этой статье описывается реализация полиморфного алгоритма о выводе типа Мильнера. Доказано, что этот алгоритм полный, эффективний и поддерживает богатую и гибкую типовую систему. Для этой цели использован язык программирования с равенствами и унификацией W.

*Magdalina Todorova, Dimitar Birov.* ALGORITHM OF CONSTRUCTION OF THE TYPE AND ITS REALIZATION IN THE LANGUAGE W

The notion of polymorphism describes data, which possess more than one type, or functions, which operate over a set of types in programming languages. In this paper an implementation of Milner's polymorphic type interface algorithm is described. It is proved that the algorithm is sound, effective and supports a rich and flexible type system. A programming language with equations and unification W is used.

## 1. ПОЛИМОРФИЗЪМ

Понятието *полиморфизъм* [7] в езиците за програмиране характеризира данни, които имат повече от един тип, или функции, които оперират върху множество типове. Полиморфизмът е следствие от взаимодействието на две взаимно противоположни цели в проектирането на езици за

програмиране — статичното типизиране на програмите и многократното им използване.

Статичното типизиране като основна характеристика на типовата система отразява способност да се локализират класове от грешки в програмите. Анализът на текста на една програма, който се извършва в процеса на компилация, дава информация за грешки, които биха могли да възникнат по време на нейното изпълнение. Ето защо статичното типизиране е основен инструмент при изграждане на високо структурирани и надеждни софтуерни системи.

Многократното използване е възможност една и съща програма да се използва в безкраен брой приложения. Характерен пример за това е прилагането на готова програма към нови типове данни. Дефинираните по такъв начин абстракции [6] предлагат изграждане на добре структурирани програмни системи.

Статичното типизиране усложнява многократната употреба на програмата. От друга страна, е трудно да се извърши статична проверка на типа на програми, предназначени за многократна употреба. Полиморфните типови системи са успешен опит за съчетаване на сигурността, която предлагат статично типизираните езици, и гъвкавостта на полиморфизма.

*Параметричният полиморфизъм* е свойство на програмите, параметризирани по отношение на типа на своите аргументи. Един език за програмиране притежава *полиморфна типова дисциплина*, ако позволява да се дефинират функции, които оперират по един и същ начин върху аргументи от различен тип.

Говори се за *явен* параметричен полиморфизъм, когато параметризирането се получава чрез явно означение на типов параметър в спецификацията на функция, както и явно прилагане на функция към типови аргументи в контекста на нейното извикване. В процеса на компилация проверката за непротиворечивост на описаната типова информация се нарича *проверка на типа*.

Когато не се допуска специфициране на формални типови параметри и прилагане на функция към такива, се говори за *неявен* параметричен полиморфизъм. В този случай типът на функцията съдържа *типови променливи*. Ако формалният параметър на функцията е типова променлива или терм, съдържащ типови променливи, тази функция може да бъде приложена към аргументи от различни типове. Тази част от процеса на компилация, в която се прави опит да се определи тип за всеки израз в програмата единствено от контекста, в който изразът се намира, се нарича *извод на тип*.

Неявният полиморфизъм може да се смята за съкратена форма на явния, ако типовите параметри и прилагането към аргумент-тип са пропуснати и трябва да бъдат изведени от езиковия процесор. Пропускането на типовите аргументи изисква да се изведе изгубената или недостигащата типова информация.

Фактически при неявния полиморфизъм се пропуска цялата типова информация. Получените в резултат на това програми се разглеждат като програми с параметри типове, означени с типови променливи. Независимо че програмите са свободни от типове, може да се извърши щателна проверка на типа.

Явният полиморфизъм е по-изразителен, понеже специфицира тип на програми, които не могат да бъдат типизирани чрез неявен полиморфизъм, но, от друга страна, е твърде многословен. На практика обаче дори в явния полиморфизъм се налага част от типовата информация да бъде пропусната и това създава бяло поле между пълния явен и пълния неявен полиморфизъм. В това бяло поле техниките за извод на тип за неявен полиморфизъм могат да бъдат полезни и това е една добра причина за изучаване на неявния полиморфизъм дори в контекста на явен полиморфизъм. Като пример на един добър и смислен компромис може да се приеме явно специфициране на типовия параметър при деклариране на функция, но след това да се използва неявният стил при прилагане на функцията, като се прилагат техниките за извод на тип за попълване на липсващата информация.

В тази статия ще опишем реализация на полиморфния алгоритъм за извод на тип на Milner [7], като използваме езика за програмиране с равенства и унификация W. Доказано е, че този алгоритъм е пълен, ефективен и поддържа богата и гъвкава типова система [7].

Пълнотата е свойство на алгоритъма за проверка (извод) на тип, което гарантира надеждност на програмата по време на изпълнение. Ако една програма премине успешно през проверката на тип, то е сигурно, че няма да възникне типова грешка по време на нейното изпълнение. Това дава възможност изпълнението на програмата да се разтовари от проверките за типови грешки, което позволява ефективна реализация на конкретния език за програмиране [11, гл. 8].

Въпреки прогреса, който се отбелязва в полиморфните езици, една характеристика остава уникална за алгоритъма на Milner — възможността да се изведе тип при липса на декларации на типове в програмите. Като предполага работа с програми, опериращи върху много типове, този алгоритъм определя *най-общия*, почти абстрактен тип на програмата. Декларацията на типа в този случай може да дефинира конкретен екземпляр на този тип и с това да ограничи областта му на приложимост.

Това свойство прави алгоритъма на Milner практически подходящ за приложение в езици, позволяващи интерактивен режим на работа. Потребителят рядко специфицира информация за типа на въвежданите от него данни. Тя може автоматично да бъде изведена и проверена.

Съвременните езици за програмиране — като (Standard) ML [8], Miranda* [17, 18], Haskell [5], комбинират сигурността на статичната проверка на типа с гъвкавостта на програмирането без декларации на типове.

* Miranda е запазена марка на Research Software Limited.

## 2. АЛГОРИТЪМ ЗА ИЗВОД НА ТИП В ПОЛИМОРФНИ ТИПОВИ СИСТЕМИ

Milner [7] реализира първия алгоритъм за извод на тип и показва, че за всеки терм, образуван съгласно класическия език за $\lambda$-смятане, този алгоритъм извежда тип. По-късно Damas и Milner [3] доказват принципното типово свойство за езици, базиращи се на разширено $\lambda$-смятане, което показва, че типовите системи са разрешими, т. е. съществува рекурсивна процедура, която за всеки терм определя тип.

В този раздел ще следваме общоприетата нотация. За да опишем алгоритъма, ще дефинираме елементарен апликативен език — разширено $\lambda$-смятане, свободно от типова информация. Счита се, че то обхваща и моделира основните изчислителни характеристики на съвременните езици за програмиране. За целите на проверка и извод на тип класическото $\lambda$-смятане е разширено с две допълнителни конструкции, даващи възможност:

- да се дефинират променливи (израза `let`) и
- за специфициране на рекурсивни дефиниции (оператора `fix`).

По отношение извода на тип първата конструкция позволява да се въведе терминът *родова* променлива, който е съществен в алгоритъма за извод на тип, а разширяване на правилото за извод на тип за втората позволява да се дефинира по-гъвкава и изразителна типова система.

Формално третиране на тези проблеми е представено в [7, 10].

### 2.1. ЕЛЕМЕНТАРЕН АПЛИКАТИВЕН ЕЗИК

За да изследваме извода на полимофен тип, ще се ограничим в нотационно минимален програмен език — *разширено $\lambda$-смятане* [1], свободно от типове. За целта ще използваме стандартните означения [7, 10]. Този език представлява ядро на модела, описващ конкретен език за програмиране. Той включва $\lambda$-абстракция и прилагане на функция, както и рекурсивни и нерекурсивни дефиниции. Разглежданията ще направим върху чистото $\lambda$-смятане без константи, тъй като константите могат да се разглеждат като променливи с конкретни връзки в глобална среда.

Множеството $\Lambda$ от *$\lambda$-изрази* (*термове*) се дефинира от следната абстрактна граматика:

$$e ::= x \mid \lambda x.e \mid \texttt{let}\ x = e\ \texttt{in}\ e' \mid \texttt{fix}\ x.e \mid \texttt{if}\ e\ \texttt{then}\ e'\ \texttt{else}\ e'',$$

където $x$ се изменя в изброимо безкрайно множество $\mathbf{V}$ от *променливи*.

Операционната семантика се дефинира чрез *редукция*, която е рефлексивна, транзитивна и съвместима обвивка на *понятието за редукция* [1, гл. 3] $\rightarrow$, дефинирана чрез

$$(\lambda x.e)e' \rightarrow e[e'/x]$$
$$\texttt{let}\ x = e'\ \texttt{in}\ e \rightarrow e[e'/x]$$
$$\texttt{fix}\ x.e \rightarrow e[\texttt{fix}\ x.e/x].$$

$e[e'/x]$ означава израза, който се получава като резултат от замяната на $e'$ за всички срещания на $x$ в $e$. Това показва, че термовете **let** и **fix** в безтиповото $\lambda$-смятане се изразяват чрез $\lambda$-абстракция и апликация. За целите на извода на тип тези изрази не са общо типизируеми в типовите дисциплини. Ето защо ние ще изучаваме двете форми, които са въведени като отделни езикови примитиви.

## 2.2. ТИПОВЕ И ТИПОВИ ПРИМИТИВИ

Тип може да бъде типова променлива или *типов оператор*. Типов оператор, който служи за конструиране на стойности, се нарича *типов конструктор*. Типовите конструктори **Nat** — за конструиране на естествените числа, и **Bool** — за конструиране на булевите стойности, са типови оператори с нулева размерност. Параметричният типов оператор $\rightarrow$ е бинарен типов конструктор, който от двата си аргумента $\alpha$ и $\beta$, означаващи произволни типове, конструира нов функционален тип $\alpha \rightarrow \beta$. Тъй като операторът $\rightarrow$ е дясно асоциативен, то означението $\alpha \rightarrow \beta \rightarrow \gamma$ заменя $\alpha \rightarrow (\beta \rightarrow \gamma)$.

Нека $\mathbf{TV}$ е безкрайно множество от *типови променливи*, така че $\mathbf{TV} \cap \mathbf{V} = \varnothing$. *Типовите изрази* се формират от следната абстрактна, контекстно-свободна граматика:

(1) $$\tau ::= \alpha \mid \tau \rightarrow \tau$$

(2) $$\sigma ::= \tau \mid \forall\alpha.\sigma,$$

където $\alpha$ се изменя в множеството $\mathbf{TV}$, а кванторът за общност $\forall$ е оператор, който свързва типовите променливи. В такъв случай имаме естествена концепция за *свободни* и *свързани* типови променливи.

Типовите изрази, които се получават от (1), се наричат (*прости*) *типове*, а по-голямото множество от типови изрази, което се получава от (2), се нарича *типови схеми*. Една типова схема се нарича *затворена*, ако не съдържа свободни типови променливи. Ще отбележим, че типовите схеми са квантувани само на най-външно ниво [10].

Типове, които съдържат типови променливи, се наричат *полиморфни*, а типовете, които не съдържат типови променливи, се наричат *мономорфни*.

Полиморфният обект може да приема различни типове при различните си срещания, където тези типове са екземпляри, получени чрез *субституция* на типовите променливи от типовата схема на обекта.

Съществуват два основни подхода към формалната семантика на типовете. Единият е дефиниране на математически модели за типовете чрез изобразяване на всеки типов израз в множество от стойности, които принадлежат на този тип, а другият — дефиниране на формални системи от аксиоми и правила за извод, в които е възможно да се докаже, че един израз притежава някакъв тип.

Обикновено семантичният модел се използва като ръководство при дефиниране на формална система за извод на типа. Всяка формална система трябва да бъде вътрешно непротиворечива, което най-често се показва чрез предлагане на модел за това.

## 2.3. ПРЕДИКАТНИ ПОЛИМОРФНИ СИСТЕМИ ЗА ИЗВОД НА ТИП

Типовите дисциплини се определят от системи за извод върху типовите твърдения за $\lambda$-изразите. В семантиката на типовете са известни два стила на типизиране:

- дескриптивен — известен като стил на Curry;
- декларативен — известен като стил на Church.

Дескриптивният стил предполага дефиниране на програми независимо от типовете. Типизирането *описва* свойствата на такива програми.

В декларативния стил програмите се дефинират, като се обявяват предварително типовете. В този случай типовете *предписват* какво представляват *добре типизираните* програми.

За извода на тип тези два стила не се различават съществено. Изтриването на типовата информация в декларативния стил води до получаване на програма в дескриптивен стил. Ако получената програма е добре типизирана, същото може да се твърди и за програмата, от която тя е получена. Тази техника е широко разпространена в теорията на типовете и изследването на свойствата на типовите системи.

Експозиция за приложимостта на типовата теория в дизайна на езици за програмиране представят Cardelli и Wegner [2]. Задълбочени трактати върху $\lambda$-смятането са работите на Barendregt [1] и Hindley и Seldin [4]. Mitchell [9] прави преглед на семантиката на простото типизирано $\lambda$-смятане.

Поради синтактичната си природа проверката на типа е по-стриктно свързана с формалните системи, отколкото с моделите. В определен смисъл алгоритъмът за проверка на тип реализира формалната система чрез осигуряване на процедура за доказване на теореми в тази система. Формалната система е съществено по-проста и по-фундаментална, отколкото какъвто и да било алгоритъм, така че най-простото представяне на алгоритъм за проверка на тип е самата формална система за извод, която той реализира. Когато търсим алгоритъм, по-добре е да се дефинира първо формалната система за него. Не всички формални типови системи позволяват алгоритъм за проверка на тип. Ако формалната система е твърде мощна (т. е. позволява да бъдат доказани много неща в нея), то е възможно тя да е неразрешима (като например системата на Mycroft). В този случай не може да се намери процедура за решението. Проверката на типа обикновено се отнася до разрешимите типови системи, за които могат да бъдат конструирани алгоритми за проверка на тип.

Съществуват два различни подхода към проверката на типа: единият е *решаване на система от типови равенства*, а другият — *доказване на*

*теореми във формална система.* Тези два подхода са взаимно заменяеми, но вторият като че ли осигурява поглед отвътре поради връзката си с типовата семантика, от една страна, и с алгоритмите, от друга.

## 2.4. ИЗВОД НА ПОЛИМОРФЕН ТИП

Системата за извод на тип, представена по-долу, се състои от една аксиома (var) и шест правила за извод на тип.

Типовото предположение $A$ е списък от двойки $x : \sigma$, където всяко $x$ се среща точно веднъж, а $\sigma$ е типова схема, която се асоциира с $x$ в типовото предположение $A$.

Нотацията $A \vdash e : \sigma$ означава, че от даденото множество $A$ от предположения за типа на променливите в него може да се докаже, че $\lambda$-изразът $e$ има тип $\sigma$. Хоризонталната черта в правилата за извод се чете *следва*.

Системата за извод описва най-малката тричленна релация $\vdash$, затворена относно следните правила:

(var) $A.x : \sigma \vdash x : \sigma$

(→ I) $\dfrac{A.x : \tau' \vdash e : \tau}{A \vdash \lambda x.e : \tau' \to \tau}$ (→ E) $\dfrac{A \vdash e : \tau' \to \tau \quad A \vdash e' : \tau'}{A \vdash (ee') : \tau}$

(∀I) $\dfrac{A \vdash e : \sigma \quad \alpha \notin \mathrm{FTV}(A)}{A \vdash e : \forall\alpha.\sigma}$ (∀E) $\dfrac{A \vdash e : \forall\alpha.\sigma}{A \vdash e : \sigma[\tau/\alpha]}$

(let) $\dfrac{A \vdash e : \sigma \quad A.x : \sigma \vdash e' : \sigma'}{A \vdash \mathbf{let}\ x = e\ \mathbf{in}\ e' : \sigma'}$ (fix) $\dfrac{A.x : \tau \vdash e : \tau}{A \vdash \mathbf{fix}\ x.e : \tau}$

(if) $\dfrac{A \vdash e : \mathbf{bool} \quad A \vdash e' : \sigma \quad A \vdash e'' : \sigma}{A \vdash \mathbf{if}\ e\ \mathbf{then}\ e'\ \mathbf{else}\ e'' : \sigma}$

*Пример* 1. Нека да демонстрираме извода на тип на функцията $\mathrm{Id} = \lambda x.x$.

$$\dfrac{\dfrac{x : \alpha \vdash x : \alpha \qquad (\mathrm{var})}{\vdash \lambda x.x : \alpha \to \alpha \qquad (\to \mathrm{I})}}{\vdash \lambda x.x : \forall\alpha.\alpha \to \alpha \quad (\forall \mathrm{I})}$$

Специализиран тип за функцията $\mathrm{Id}(5)$ се извежда от общия тип:

$$\dfrac{\vdash \lambda x.x : \forall\alpha.\alpha \to \alpha \qquad (\forall \mathrm{E})}{\vdash \lambda x.x : \mathbf{Int} \to \mathbf{Int}}$$

*Пример* 2. Нека дефинираме функцията от по-висок ред **map**, която прилага функция върху всеки от елементите на линеен списък:

$\mathbf{map}\, f\, [\,] = [\,]$
$\mathbf{map}\, f\, (x{::}xs) = ((f\, x){::}(\mathbf{map}\, f\, xs))$,

където с [] означаваме празния списък.

Ще демонстрираме извода на тип за функцията **map**. За да изведем тип за функцията **map**, са необходими следните аксиоми, отнасящи се до типовия конструктор *списък*, който ще означаваме $[]$.

(NIL) $\quad \vdash [] : [\alpha]$

За конструктора на списъци, означаван като ::, имаме следната аксиома:

(CONS) $\quad \vdash :: : \alpha \to [\alpha] \to [\alpha]$

С цел да демонстрираме процеса на дедукция на типа на функцията **map** ще транслираме горните две равенства в $\lambda$-израз. Тъй като тя е рекурсивна функция (от по-висок ред), ще използваме оператора **fix**:

$$\texttt{fix map}.\lambda f.\lambda l.\texttt{if null}(l)\ \texttt{then}\ [] \\ \texttt{else}\ (f(\texttt{hd}(l))::(\texttt{map}\ f\ \texttt{tl}(l)))$$

В горния терм се съдържат три помощни функции — **null**, **hd** и **tl**, които оперират със списъци и за тях са в сила следните аксиоми:

(NULL) $\vdash \texttt{null} : [\alpha] \to \texttt{bool}$,
(HD) $\vdash \texttt{hd} : [\alpha] \to \alpha$,
(TL) $\vdash \texttt{tl} : [\alpha] \to [\alpha]$,

където **bool** е типов оператор (с нулева размерност — типова константа). Ще демонстрираме процеса на извод на тип на стъпки. Той започва отвътре (най-вътрешните $\lambda$-изрази) навън:

$$\frac{l : \gamma \vdash l : \gamma \quad (\text{var})}{l : \gamma \vdash l : \forall\gamma.\gamma \quad (\forall\text{I})}$$

$$\frac{l : [\alpha] \vdash l : [\alpha] \quad (\forall\text{E}) \qquad \texttt{hd} : [\alpha] \to \alpha \vdash \texttt{hd} : \forall\alpha.[\alpha] \to \alpha \quad (\forall\text{I})}{l : [\alpha] \vdash \texttt{hd}(l) : \alpha \qquad (\to \text{E})}$$

$$f : \delta \vdash f : \delta \quad (\text{var}) \qquad f : \delta \vdash f : \forall\delta.\delta \quad (\forall\text{I}) \qquad f : \alpha \to \beta \vdash f : \alpha \to \beta \quad (\forall\text{E})$$

$$\frac{l : [\alpha] \vdash \texttt{hd}(l) : \alpha}{l : [\alpha], f : \alpha \to \beta \vdash f(\texttt{hd}(l)) : \beta \quad (\to \text{E})}$$

Вече имаме две двойки променлива и тип, съответстващи на свързаните с $\lambda$ променливи. Типовото предположение се запазва същото и в следващите дедукции:

$$\frac{l : [\alpha] \vdash l : [\alpha] \quad (\text{var}) \qquad \texttt{tl} : [\alpha] \to [\alpha] \vdash \texttt{tl} : \forall\alpha.[\alpha] \to [\alpha] \quad (\forall\text{I})}{l : [\alpha] \vdash \texttt{tl}(l) : [\alpha] \quad (\to \text{E})}$$

$$\frac{\texttt{map} : \chi \vdash \texttt{map} : \chi \quad (\text{var}) \qquad \texttt{map} : \chi \vdash \texttt{map} : \forall\chi.\chi \quad (\forall\text{I})}{\dfrac{\texttt{map} : \forall\chi.\chi \vdash \texttt{map} : (\alpha \to \beta) \to \delta \quad (\forall\text{E})}{f : \alpha \to \beta, \texttt{map} : (\alpha \to \beta) \to \delta \vdash \texttt{map}f : \delta \quad (\to \text{E})}}$$

$$\frac{\texttt{map}f : \delta \vdash \texttt{map}f : \forall\delta.\delta \quad (\forall\text{I}) \qquad \texttt{map}f : \forall\delta.\delta \vdash \texttt{map}f : [\alpha] \to \xi \quad (\forall\text{E})}{l : [\alpha], f : \alpha \to \beta, \texttt{map} : (\alpha \to \beta) \to [\alpha] \to \xi \vdash \texttt{map}f : [\alpha] \to \xi}$$

$$\frac{l : [\alpha] \vdash \texttt{tl}(l) : [\alpha]}{l : [\alpha], f : \alpha \to \beta, \texttt{map} : (\alpha \to \beta) \to [\alpha] \to \xi \vdash (\texttt{map}f\texttt{tl}(l)) : \xi \quad (\to \text{E})}$$

$$\frac{l : [\alpha], f : \alpha \to \beta, \texttt{map} : (\alpha \to \beta) \to [\alpha] \to \xi \vdash (\texttt{map}f\texttt{tl}(l)) : \forall\xi.\xi \quad (\forall\text{I})}{l : [\alpha], f : \alpha \to \beta, \texttt{map} : (\alpha \to \beta) \to [\alpha] \to [\beta] \vdash (\texttt{map}f\texttt{tl}(l)) : [\beta] \quad (\forall\text{E})}$$

$$l : [\alpha], f : \alpha \to \beta \vdash f(\mathtt{hd}(l)) : \beta \qquad \vdash (::) : \beta \to [\beta] \to [\beta] \quad \text{(CONS)}$$

Нека с $A$ означим $\{l : [\alpha], f : \alpha \to \beta, \mathtt{map} : (\alpha \to \beta) \to [\alpha] \to [\beta]\}$. Тогава

$$A \vdash (f(\mathtt{hd}(l))::(\mathtt{map} f \mathtt{tl}(l))) : [\beta] \quad (\to \text{E})$$

$$\frac{A \vdash \mathtt{null} : [\alpha] \to \mathtt{bool} \qquad A \vdash l : [\alpha] \quad \text{(var)}}{A \vdash \mathtt{null}(l) : \mathtt{bool} \qquad (\to \text{E})}$$

$$\frac{\vdash [] : [\beta] \quad \text{(NIL)}}{A \vdash \mathtt{if}\ \mathtt{null}(l)\ \mathtt{then}\ []\ \mathtt{else}\ (f(\mathtt{hd}(l))::(\mathtt{map} f \mathtt{tl}(l))) : [\beta] \quad \text{(IF)}}$$

$$A \vdash l : [\alpha] \quad \text{(var)}$$

Нека означим с $M$ израза $\mathtt{if}\ \mathtt{null}(l)\ \mathtt{then}\ []\ \mathtt{else}\ (f(\mathtt{hd}(l))::(\mathtt{map} f \mathtt{tl}(l)))$. Тогава

$$\frac{\dfrac{f : \alpha \to \beta, \mathtt{map} : (\alpha \to \beta) \to [\alpha] \to [\beta] \vdash \lambda l.M : [\alpha] \to [\beta] \quad (\to \text{I}) \qquad f : \alpha \to \beta, \mathtt{map} : (\alpha \to \beta) \to [\alpha] \to [\beta] \vdash f : \alpha \to \beta \quad \text{(var)}}{\mathtt{map} : (\alpha \to \beta) \to [\alpha] \to [\beta] \vdash \lambda f.\lambda l.M : (\alpha \to \beta) \to [\alpha] \to [\beta] \quad (\to \text{I})}}{\vdash \mathtt{fix}\ \mathtt{map}.\lambda f.\lambda l.M : (\alpha \to \beta) \to [\alpha] \to [\beta] \quad \text{(FIX)}}$$

## 2.5. АЛГОРИТЪМ ЗА ИЗВОД НА ПОЛИМОРФЕН ТИП

Изводът на типа като част от единен процес може да определи дали програмата е добре типизирана и ако това е така, да изведе тип за всеки израз в нея.

Израз за формата на типа на даден терм се получава, като се изследва контекстът, в който този терм участва. Полученият типов израз най-добре се вгражда в този контекст. Като се изследва същият терм сам за себе си, се извежда форма за типа на този терм, която той може да приеме. Тази форма може да бъде достатъчно обща. Така получените два типови израза обикновено съдържат променливи. Първият израз е форма на типа, изисквана от контекста (тип на израза, изведен „отвън“), а вторият е форма на типа, който обектът може да приеме (тип, изведен „отвътре“). За да бъде добре типизиран даден терм, тези два типови израза трябва да си съответстват. Съответствието между тях се постига чрез съпоставяне на двата типови израза. Това означава, че съществува субституция, която действа върху родовите променливи в двата израза, такава че те могат да бъдат доведени до една и съща форма (унифицирани).

В този раздел ще опишем алгоритъма за извод на тип на Milner [7], който за дадено типово предположение $A$ и терм $e$ намира субституция $S$ и тип $\tau$, такива че $SA \vdash e : \tau$. Двойката $S$ и $\tau$ е най-общата такава двойка. Ако не съществува такава двойка, алгоритъмът прекъсва работа и издава съобщение за лошо типизиране в програмата.

В реализацията на алгоритъма се използва лемата на Robinson [12]:

**Лема 1.** *Съществува алгоритъм* $U : \tau \times \tau \to S + \{\mathtt{fail}\}$, *където* $\tau$ *е множество от типове, а* $S$ *— субституция, такъв че*:

(i) *Ако* $U(\tau_1, \tau_2) =$ **fail**, *то не съществува субституция* $S$, *за която* $S\tau_1 = S\tau_2$.

(ii) *Ако* $U(\tau_1, \tau_2) = S$, *то* $S\tau_1 = S\tau_2$ *и всяка друга субституция* $S'$ *с това свойство може да бъде изразена чрез* $S' = RS$ *за някаква субституция* $R$.

Нещо повече, субституцията $S$, която се получава от алгоритъма за унификация, е идемпотентна и действа само върху променливите на $\tau_1$ и $\tau_2$.

Ще дефинираме алгоритъма W на Milner—Mycroft за извод на тип.

**Алгоритъм за извод на тип** $\mathrm{W}(A, e)$

Нека $e$ има вида:

1. $x$. Ако $x : \sigma$, където $\sigma = \forall\alpha.\tau$, то $(S = \mathrm{Id}, \beta/\alpha)$, където Id е субституцията идентитет, а за всяка родова променлива $\alpha$ в типовата схема $\sigma$ се въвежда нова типова променлива $\beta$. В противен случай резултатът от действието на алгоритъма е fail (неуспех).

2. $\lambda x.e$. Нека $(S_1, \tau_1) = \mathrm{W}(A\{x : \beta\}, e)$, където $\beta$ е нова типова променлива в $(S_1, S_1\beta \to \tau_1)$.

Ако променливата $x$ е свързана с $\lambda$-абстракция $(\lambda x.e)$, се въвежда нова типова променлива $\alpha$. Конкретният вид на типа, означен с $\alpha$, ще се определи от контекста, в който променливата $x$ се среща. Двойката $(x : \alpha)$ се поставя в средата, която се претърсва всеки път, когато се срещне променливата $x$ в контекста.

В $\lambda$-абстракцията типът на израза $e$ се извежда в контекста, в който на променливата $x$ е присвоена нова типова променлива $\alpha$.

3. $(ee')$. Нека $(S_1, \tau_1) = \mathrm{W}(A, e)$, $(S_2, \tau_2) = \mathrm{W}(S_1A, e')$, $V = U(S_2\tau_1, \tau_2 \to \beta)$, където $\beta$ е нова типова променлива в $(VS_2S_1, V\beta)$.

В прилагането на израза $e$ към израза $e'$ типът на израза $e$ трябва да бъде от вида $\alpha \to \beta$, където $\alpha$ е типът на израза $e'$, а $\beta$ е нова типова променлива. Типът на $e$ е функционален тип, а типът, асоцииран с $\beta$, е типът на целия израз.

4. **let** $x = e$ **in** $e'$. Нека $(S_1, \tau_1) = \mathrm{W}(A, e)$, $A_1 = (S_1A)\{x : S_1A(\tau_1)\}$, $(S_2, \tau_2) = \mathrm{W}(A_1, e')$ във $(S_2S_1, \tau_2)$.

5. **fix** $x.e$. Нека $\sigma_0 = \forall\beta.\beta$, където $\beta$ е някаква типова променлива, и $A_0 = A\{x : \sigma_0\}$. Тогава повтаряй $(S_{i+1}, \tau_{i+1}) = \mathrm{W}(A_i, e)$ за $i \geq 0$, $\sigma_{i+1} = S_{i+1}A_i(\tau_{i+1})$, $A_{i+1} = (S_{i+1}A_i)\{x : \sigma_{i+1}\}$, докато $S_{i+1}\sigma_i = \sigma_{i+1}$ в $(S_{i+1} \ldots S_2S_1, \tau_{i+1})$.

6. **if** $e$ **then** $e'$ **else** $e''$. Нека $(S_0, \tau_0) = \mathrm{W}(A, e)$ и $V = U(\tau_0, \mathbf{bool})$, $(S, \tau) = \mathrm{W}(VS_0A, e')$ и $(S_1, \tau_1) = \mathrm{W}(SVS_0A, e'')$.

Нека $V_1 = U(S_1\tau, \tau_1)$, тогава резултатът е $(V_1S_1SVS_0, \tau)$.

**Лема 2** ((синтактична) необходимост и достатъчност на W за релацията $\vdash$). *За дадени* $A$ *и* $e$ *имаме*:

(i) *Ако алгоритъмът* $\mathrm{W}(A, e)$ *приключи работа успешно и се получи като резултат двойката* $(S, \tau)$, *то* $SA \vdash e : \tau$.

(ii) *Ако за субституция* $S'$ *и тип* $\tau$ *имаме, че* $S'A \vdash e : \sigma$, *то*:
a) $\mathrm{W}(A, e)$ *приключва успешно работа с резултат* $(S, \tau)$ *и*
b) $S'A = RSA$ *и* $R(SA(\tau)) \subseteq \sigma$ *за някоя субституция* $R$.

*Доказателство.* Вж. [10].

## 3. ЕЗИК ЗА ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ С УНИФИКАЦИЯ W

Езикът W реализира един подход за интегриране на функционалното и логическото програмиране [11]. Той е език за програмиране с равенства, обобщен в следните две направления:

- функционалните термове се заменят с думи от КС-език;
- функционалната оценка се заменя с логическа.

Тъй като при стила за програмиране с равенства функция се дефинира с помощта на едно или повече равенства, то, най-общо казано, програмата на езика W е система от равенства. Докато обикновените интерпретатори на равенства действат като трансформатори на функционални термове, програмите на езика W могат да описват трансформации на низове, а не само на функционални термове. Структурата на тези низове се описва с помощта на КС-граматика. Известните интерпретатори на равенства оценяват само термове, които не съдържат променливи. В езика W е възможно да се извършва не само функционално, а и логическо оценяване на думи от КС-език. Затова W се нарича още език за програмиране с равенства в логически стил.

Програмите на езика W съдържат синтактични дефиниции и равенства. Синтактичните дефиниции задават КС-граматика, определяща езика, чиито думи ще бъдат оценявани. Равенствата задават правилата, по които едни низове се трансформират в други. Всички променливи, използвани в равенствата, се декларират в раздела на променливите. Променливи могат да се генерират от интерпретатора на езика в процеса на оценяване. Оценяването в езика W се осъществява чрез прилагане на стратегията „най-ляво и най-външно заместване“. Резултатът от оценката се състои от една или повече думи от дефинираната КС-граматика и може да включва субституции за променливите от входната дума. Подробно описание на синтаксиса и семантиката на езика W е направено в [13, 15]. Примери за някои приложения на езика са дадени в [13, 16].

Изходът на програмите, написани на езика W, представлява една или няколко основни или неосновни думи. Съществено е, че изходът може да включва някои субституции на променливите в изходната дума.

Програмите на езика W се наричат обобщени трансформатори. Те са описани и математически обосновани в [14, 15].

Системите за извод на тип са дедуктивни системи, в които въз основа на формални правила за извод на тип се доказва определено свойство на

израз (терм). Типът в системите на Milner—Mycroft се разглежда като предикат, който удовлетворява субекта — $\lambda$-израза. По същество изводът на типа представлява последователно прилагане на правилата от системата за извод, за да се докаже или отхвърли валидността на предикат за разглеждания $\lambda$-израз. Това е процес на доказване на теореми, за което езикът W се оказва подходящ [16].

Широко използваният алгоритъм на Milner [7] за определяне на типа на $\lambda$-терм се основава на принципа на резолюцията на Robinson [12]. Езикът W се базира на унификацията [13, 15], което определи неговия избор като език за реализация.

### 3.1. СИНТАКСИС НА ЕЗИКА W

Фиг. 1 илюстрира програма на езика W [16]. Програмата дава възможност за събиране, умножение и сравняване за равенство на естествени числа.

```
Transformer NatNum : ⟨T⟩;
  type
    ⟨T⟩          ::= ⟨nat⟩ | ⟨bool⟩;
    ⟨nat⟩        ::= ⟨func_name⟩(⟨nat⟩, ⟨nat⟩) | s(⟨nat⟩) | 0;
    ⟨bool⟩       ::= eq(⟨nat⟩, ⟨nat⟩) | true | false;
    ⟨func_name⟩  ::= + | *;
  var
    ⟨x⟩, ⟨y⟩ : ⟨nat⟩;
    ⟨f⟩, ⟨g⟩ : ⟨func_name⟩;
begin
{T1}  +(0, ⟨x⟩)          ⟹ ⟨x⟩;
{T2}  +(s(⟨x⟩), ⟨y⟩)     ⟹ s(+(⟨x⟩, ⟨y⟩));
{T3}  *(0, ⟨x⟩)          ⟹ 0;
{T4}  *(s(⟨x⟩), ⟨y⟩)     ⟹ +(*(⟨x⟩, ⟨y⟩), ⟨y⟩);
{T5}  eq(0, 0)           ⟹ true;
{T6}  eq(s(⟨x⟩), 0)      ⟹ false;
{T7}  eq(0, s(⟨x⟩))      ⟹ false;
{T8}  eq(s(⟨x⟩), s(⟨y⟩)) ⟹ eq(⟨x⟩, ⟨y⟩)
end.
```

Фиг. 1

Някои линии на програмата са номерирани с цел да се илюстрира изпълнението ѝ. От примера се вижда, че програмата на езика W се състои от три части:

- описание на типовете данни,
- описание на променливите,
- раздел на трансформациите.

Типовете данни, които се обработват от програмите на езика W, се описват чрез версия на метаезика на Бекус—Наур. Трансформациите са

правила, които описват преобразуването на едни низове в други. Те могат да съдържат променливи, които се описват в раздела на променливите.

Синтаксисът на програма на езика W е следният:

⟨program⟩ ::= ⟨heading⟩⟨block⟩
⟨heading⟩ ::= **Transformer** ⟨name⟩ : ⟨main_type⟩
⟨block⟩ ::= ⟨types_declaration⟩
⟨variables_declaration⟩
⟨transformations_section⟩.

**3.1.1. Типове данни в езика W.** Описанието на типовете данни в езика W се извършва по начин, подобен на езика на Бекус—Наур. Езикът допуска три типа данни: дефинирани, примитивни и съставни.

Дефинираните типове съответстват на метапроменливите, примитивните типове — на терминалните символи, а съставните типове — на редица от метапроменливи и терминални символи от метаезика на Бекус—Наур.

Дефинираните типове се означават чрез име, поставено в ъгловите скобки ⟨ и ⟩. В програмата NatNum дефинирани типове са ⟨**T**⟩, ⟨**nat**⟩, ⟨**bool**⟩ и ⟨**func_name**⟩. Множеството от стойности на дефинирания тип е равно на обединението от множествата от стойности на типовете, участващи в неговата дефиниция.

Примитивните типове се означават като редици от символи, несъдържащи и неоградени със скобите ⟨ и ⟩. Множеството от стойности на примитивния тип е множество от един елемент — самата редица от символи. Например **s**(, ), +, *, **eq**( са някои от примитивните типове, използвани от програмата NatNum.

Съставният тип се дефинира като редица от два или повече други типа. Например **eq**(⟨**nat**⟩, ⟨**nat**⟩) и ⟨**func_name**⟩(⟨**nat**⟩, ⟨**nat**⟩) са съставни типове на програмата NatNum.

Описанието на типовете данни се извършва в раздела на типовете. Синтаксисът на този раздел е следният:

⟨type_declaration⟩::= **type** ⟨type_definition⟩;
{⟨type_definition⟩;}
⟨type_definition⟩::= ⟨defined_type_name⟩::= ⟨defining_type⟩
{|⟨defining_type⟩},

където ⟨defining_type⟩ е произволен тип. Символът вертикална черта се използва като металингвистичен съюз и означава „или“.

Разделът на типовете дефинира КС-граматика, чиято аксиома е ⟨main_type⟩. Тази граматика дефинира синтаксиса на изразите, които се оценяват от програмите на езика W.

**3.1.2. Описание на променливи.** Всяка променлива, използвана в раздела на трансформациите, а също и в оценявания израз, трябва да се опише в раздела на променливите. Този раздел има следния синтаксис:

⟨variable_declaration⟩:: = ⟨empty⟩ |
**var** ⟨variables_description⟩{⟨variables_description⟩},

където

⟨variables_description⟩:: = ⟨variable_name⟩{, ⟨variable_name⟩} : ⟨type⟩;

Всяка променлива може да получава стойност само от множеството от стойности на специфицирания ѝ тип. В този смисъл езикът W е силно типизиран.

**3.1.3. Трансформации.** Трансформациите са правила, които преобразуват едни низове в други. Реализират се като равенства. Синтаксисът на раздела на трансформациите е следният:

⟨transformations_section⟩:: =
**begin**
⟨transformation⟩
{; ⟨transformation⟩}
**end**,

където

⟨transformation⟩:: = ⟨TypeInfer⟩ ⟹ ⟨TypeInfer⟩.

⟨TypeInfer⟩ е редица от примитивни типове и променливи, конструирани в съответствие със синтаксиса, дефиниран в раздела на типовете и раздела на променливите. Ще я наричаме шаблон или образец.

**3.1.4. Изрази.** Изразите в езика W са редици от примитивни типове и променливи, генерирани в съответствие със синтаксиса, дефиниран в раздела на типовете и раздела на променливите. Програмите на езика W оценяват изразите чрез подходящи субституции на променливите. За тази цел синтаксисът на изразите се разширява чрез добавяне на субституции към обикновените изрази, т. е.

⟨expression⟩:: = ⟨TypeInfer⟩{ **if** ⟨substitution⟩},

където

⟨substitution⟩:: = (⟨variable_name⟩ = ⟨TypeInfer⟩
{⟨variable_name⟩ = ⟨TypeInfer⟩}) | ⟨empty⟩.

Програмите на езика W се прилагат към изрази с празни субституции. В резултат се получава списък от изрази.

От направените разглеждания следва, че синтаксисът на езика W е по-общ от този на езиците за програмиране с равенства. Докато конвенционалните интерпретатори на равенства действат като трансформатори на функционални термове, програмите на езика W описват трансформации на низове, генерирани от КС-граматика.

Семантиката на езика W ще разгледаме неформално. Ще използваме програмата NatNum.

**3.2.1. Прилагане на трансформации към изрази.** Образецът на всеки израз може да се разглежда като конкатенация на три части, които ще наричаме глава, тяло и опашка. За всеки образец съществуват множество от такива представяния.

Ще разгледаме как трансформацията $U \Longrightarrow V$ се прилага към израза t if s. Възможни са следните три случая:

а) При подходящ избор на стойности на променливите на шаблона $U$ стойността на $U$ съвпада с тяло на израза t if s.

В този случай трансформацията $U \Longrightarrow V$ се прилага към израза t if s по следния начин: Субституцията, която унифицира $U$ с тяло на t, се прилага към шаблона $V$. Тялото на t се замества с получения шаблон. Субституцията s не се променя.

*Пример* 3. Нека разгледаме трансформацията

$$*(\mathbf{s}(\langle \mathrm{x} \rangle), \langle \mathrm{y} \rangle) \Longrightarrow +(*(\langle \mathrm{x} \rangle, \langle \mathrm{y} \rangle), \langle \mathrm{y} \rangle)$$

и израза $\mathbf{eq}(*(\mathbf{s}(0), \mathbf{s}(\mathbf{s}(0))), 0)$ if ( ). Ако изберем за $\langle \mathrm{x} \rangle$ и $\langle \mathrm{y} \rangle$ съответно стойностите 0 и $\mathbf{s}(\mathbf{s}(0))$, стойността на шаблона $*(\mathbf{s}(\langle \mathrm{x} \rangle), \langle \mathrm{y} \rangle)$ съвпада с тяло на дадения израз. След прилагането на трансформацията към израза се получава $\mathbf{eq}(+(*(0, \mathbf{s}(\mathbf{s}(0))), \mathbf{s}(\mathbf{s}(0))), 0)$ if ( ).

б) При подходящ избор на стойности на променливите от тяло на израза t if s стойността на тялото съвпада с шаблона $U$

В този случай трансформацията $U \Longrightarrow V$ се прилага към израза t if s по следния начин: Означаваме с s1 субституцията, която унифицира тялото на дадения израз с шаблона $U$. В израза s1(t) if s1(s) образът на тялото на t чрез s1 се замества с шаблона $V$.

*Пример* 4. Нека разгледаме трансформацията

$$\mathbf{eq}(0, 0) \Longrightarrow \mathbf{true}$$

и израза $\mathbf{eq}(\langle \mathrm{y} \rangle, 0)$ if $(\langle \mathrm{x} \rangle = \mathbf{s}(\mathbf{s}(\langle \mathrm{y} \rangle)))$. Субституцията $\mathrm{s1} = (\langle \mathrm{y} \rangle = 0)$ унифицира целия низ $\mathbf{eq}(\langle \mathrm{y} \rangle, 0)$ с шаблона $U$. В резултат от прилагането на трансформацията към израза се получава **true** if $(\langle \mathrm{x} \rangle = \mathbf{s}(\mathbf{s}(0)))$.

в) При подходящ избор на стойности за променливите от тяло на израза t if s, както и при подходящ избор на стойности на променливите на шаблона $U$, стойността на тялото съвпада със стойността на $U$.

В този случай трансформацията $U \Longrightarrow V$ се прилага към израза t if s по следния начин: Нека m и n са субституции, така че стойността на тялото след прилагане на субституцията m към израза и стойността на $U$ след прилагането на субституцията n съвпадат. В израза m(t) if m(s) образът на тялото на израза чрез m се замества с $\mathrm{n}(V)$.

*Пример* 5. Нека разгледаме трансформацията

$\mathbf{eq}(\mathbf{s}(\langle x\rangle), \mathbf{s}(\langle y\rangle)) \Longrightarrow \mathbf{eq}(\langle x\rangle, \langle y\rangle)$

и израза $\mathbf{eq}(\mathbf{s}(0), \langle z\rangle)$ if $(\langle a\rangle = \mathbf{s}(\langle z\rangle))$. Субституциите $m = (\langle z\rangle = \mathbf{s}(\langle y\rangle))$ и $n = (\langle x\rangle = 0)$ унифицират t и $U$. В резултат от прилагането се получава изразът $\mathbf{eq}(0, \langle y\rangle)$ if $(\langle a\rangle = \mathbf{s}(\mathbf{s}(\langle y\rangle)))$.

И в трите случая трансформацията и тялото на израза имат поне един общ тип, който се нарича **тип на прилагането**.

**3.2.2. Оценка на изрази чрез W-програми.** Съвкупност от трансформации, приложими към даден израз, се нарича *съвкупност от еквивалентни относно приложимост трансформации*, ако главите, телата и типовете на прилагане на всички трансформации от съвкупността са едни и същи.

Нека са дадени две съвкупности от еквивалентни относно приложимост трансформации, приложими към един и същ израз. За всяка от двете съвкупности съществуват някакви глава, тяло и тип на прилагане.

Казваме, че *първата съвкупност е приложима преди втората*, ако е изпълнено едно от следните условия:

а) Главата в случая на първата съвкупност е по-къса от главата в случая на втората.

б) Двете глави са равни, но тялото в случая на първата съвкупност е по-дълго от тялото в случая на втората.

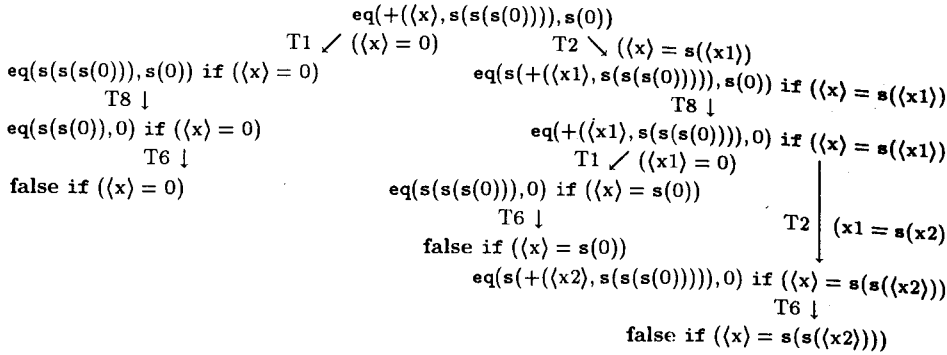
в) Главите и телата на двете съвкупности са едни и същи, но типовете им на прилагане са различни и типът на прилагане на втората съвкупност е подтип на типа на прилагане на първата съвкупност.

*Оценяването на израз чрез* W-*програма* протича на последователни стъпки. На всяка стъпка се проверява дали изразът е от главния тип. Ако не е, резултатът от оценяването е неопределен. Ако изразът е от главния тип, търсят се всички трансформации, които са приложими към него. Ако няма такива, оценката за този израз завършва. В противен случай всички приложими трансформации се разделят на съвкупности от еквивалентни относно приложимост трансформации. От всички приложими съвкупности от трансформации се избира тази, която е приложима преди останалите. Избраната съвкупност се прилага към израза. Така се получава списък от изрази. Оценяването продължава с оценяване на всеки от получените изрази чрез W-програмата.

Фиг. 2 илюстрира оценяването на израза $\mathbf{eq}(+(\langle x\rangle, \mathbf{s}(\mathbf{s}(\mathbf{s}(0)))), \mathbf{s}(0))$ чрез програмата NatNum.

Резултатът от оценката е списъкът

$$\begin{aligned}&(\text{false if } (\langle x\rangle = 0),\\ &\ \text{false if } (\langle x\rangle = \mathbf{s}(0)),\\ &\ \text{false if } (\langle x\rangle = \mathbf{s}(\mathbf{s}(\langle x2\rangle)))),\end{aligned}$$




Фиг. 2

където ⟨x2⟩ е променлива от тип ⟨nat⟩ и е генерирана от интерпретатора на езика. Тази оценка доказва, че уравнението $x + 3 = 1$ няма естествено число за корен.

Програмите на езика W наричаме обобщени трансформатори. Те са описани и математически обосновани в [14, 15]. Не всеки обобщен трансформатор осигурява подходяща програмна среда. Съществуват такива обобщени трансформатори, че за някои думи от КС-езика, породен от граматиката на трансформатора, съществуват няколко различни нормални форми. За такива обобщени трансформатори е трудно да се обоснове изборът на стратегията за оценяване. Затова се ограничаваме с разглеждането на конфлуентни обобщени трансформатори [15, 14]. При избраната стратегия на „най-ляво и най-външно заместване“ е възможно за някоя дума да съществува нормална форма, но тя да не може да се получи, тъй като процесът на оценяване е безкраен. За да не възникват такива ситуации, ще направим още едно ограничение върху обобщените трансформатори — ще искаме те да са външни [15, 14]. Свойството външност на обобщен трансформатор осигурява, че ако дума от КС-езика, породен от обобщения трансформатор, има нормална форма, тя ще бъде намерена след прилагане на стратегията на „най-ляво и най-външно заместване“. Доказано е [15], че използваната в езика W семантика осигурява пълнота, ако обобщеният трансформатор е конфлуентен и външен. Установени са също критерии [14, 15], които осигуряват обобщен трансформатор да е конфлуентен, а също и да е външен.

От направените разглеждания върху езика W можем да заключим, че оценяването на изрази реализира подход, подобен на метода на резолюцията, използван от алгоритъма за извод на тип. Освен това езикът W оперира върху КС-граматики, каквито са граматиките на типовете и типовите схеми. Системите за извод на тип са дедуктивни системи, в които въз основа на формални правила за извод на тип се доказва определено свойство на израза. Семантиката на езика W позволява чрез него да се

доказват теореми. Тези характеристики на езика W го правят подходящ за реализиране на алгоритъма за извод на тип.

## 4. РЕАЛИЗАЦИЯ НА АЛГОРИТЪМА ЗА ИЗВОД НА ТИП В ЕЗИКА W

Програмата TypeInfer реализира описания в § 2 алгоритъм на Milner.

```
Transformer TypeInfer : ⟨Bool⟩;
 type
   ⟨Bool⟩:: = P(⟨Set⟩, ⟨Lam-expr⟩ : ⟨Tip⟩) | ⟨Bool⟩ and ⟨Bool⟩ | true | false;
   ⟨Lam-expr⟩:: = ⟨V⟩ | lam ⟨V⟩.⟨Lam-expr⟩ |
                  fix ⟨V⟩.⟨Lam-expr⟩ | ⟨Lam-expr⟩⟨Lam-expr⟩ |
                  (⟨Lam-expr⟩) | let ⟨V⟩ = ⟨Lam-expr⟩ in ⟨Lam-expr⟩ |
                  if ⟨Bool⟩ then ⟨Lam-expr⟩else ⟨Lam-expr⟩;
   ⟨Set⟩:: = nil | ⟨Lam-expr⟩ : ⟨Tip⟩⟨Set⟩ |
             ⟨Set⟩⟨Lam-expr⟩ : ⟨Tip⟩ | ⟨Set⟩⟨Lam-expr⟩ : ⟨Tip⟩⟨Set⟩;
   ⟨Tip⟩:: = ⟨V⟩ | ⟨Tip⟩ → ⟨Tip⟩;
   ⟨V⟩:: = x | Bool;
 var
   ⟨a⟩, ⟨b⟩, ⟨c⟩ : ⟨Set⟩;
   ⟨e⟩, ⟨e1⟩, ⟨e2⟩, ⟨e3⟩ : ⟨Lam-expr⟩;
   ⟨f⟩ : ⟨Bool⟩;
   ⟨x⟩ : ⟨V⟩;
   ⟨t⟩, ⟨t1⟩, ⟨t2⟩, ⟨t3⟩ : ⟨Tip⟩;
begin
{T1}  P(⟨a⟩⟨x⟩ : ⟨t⟩⟨b⟩, ⟨x⟩ : ⟨t⟩) ⟹ true;
{T2}  P(⟨a⟩⟨x⟩ : ⟨t⟩⟨b⟩⟨x⟩ : ⟨t1⟩⟨c⟩, ⟨e⟩ : ⟨t2⟩) ⟹ false;
{T3}  P(⟨a⟩, lam ⟨x⟩.⟨e⟩ : ⟨t1⟩ → ⟨t⟩) ⟹ P(⟨a⟩⟨x⟩ : ⟨t1⟩, ⟨e⟩ : ⟨t⟩);
{T4}  P(⟨a⟩⟨e1⟩ : ⟨t1⟩⟨b⟩, ⟨e⟩⟨e1⟩ : ⟨t⟩) ⟹
          P(⟨a⟩⟨e1⟩ : ⟨t1⟩⟨b⟩, ⟨e⟩ : ⟨t1⟩ → ⟨t⟩) and P(⟨a⟩⟨e1⟩ : ⟨t1⟩⟨b⟩, ⟨e1⟩ : ⟨t1⟩);
{T5}  P(⟨a⟩, fix ⟨x⟩.⟨e⟩ : ⟨t⟩) ⟹ P(⟨a⟩⟨x⟩ : ⟨t⟩, ⟨e⟩ : ⟨t⟩);
{T6}  P(⟨a⟩⟨e⟩ : ⟨t⟩⟨b⟩, let ⟨x⟩ = ⟨e⟩ in ⟨e1⟩ : ⟨t1⟩) ⟹
          P(⟨a⟩⟨e⟩ : ⟨t1⟩⟨b⟩, ⟨e⟩ : ⟨t1⟩) and P(⟨a⟩⟨e⟩ : ⟨t⟩⟨b⟩.⟨x⟩ : ⟨t⟩, ⟨e1⟩ : ⟨t1⟩);
{T7}  P(⟨a⟩, if ⟨f⟩ then ⟨e1⟩ else ⟨e2⟩ : ⟨t⟩) ⟹
          P(⟨a⟩, ⟨f⟩ : Bool) and P(⟨a⟩, ⟨e1⟩ : ⟨t⟩) and P(⟨a⟩, ⟨e2⟩ : ⟨t⟩);
{T8}  P(⟨a⟩, (⟨e⟩) : ⟨t⟩) ⟹ P(⟨a⟩, ⟨e⟩ : ⟨t⟩);
{T9}  true and ⟨f⟩ ⟹ ⟨f⟩;
{T10}  ⟨f⟩ and true ⟹ ⟨f⟩;
{T11}  false and ⟨f⟩ ⟹ false;
{T12}  ⟨f⟩ and false ⟹ false;
end.
```

Множеството от $\lambda$-изрази от алгоритъма за извод на тип е дефинирано чрез типа ⟨Lam-expr⟩ на обобщения трансформатор TypeInfer. Типът ⟨Tip⟩ на обобщения трансформатор TypeInfer определя типовите изрази, а типът ⟨Set⟩ — типовото предположение. С nil е означен празният низ.

Първото равенство реализира аксиомата на системата за извод на тип. Второто равенство отразява, че типовата променлива x се среща точно веднъж в типовото предположение.

Трансформациите от трета до осма реализират правилата за извод на тип, а последните четири равенства — операцията за логическо умножение and.

Забелязваме, че следствието от правилата за извод е лява страна, а предпоставката — дясна страна на трансформациите, реализиращи правилата за извод. Означението $A \vdash e : \tau$ е реализирано чрез израза P(⟨a⟩, ⟨e⟩ : ⟨t⟩) от тип ⟨Bool⟩ и има същия смисъл. За програмата TypeInfer са в сила условията за конфлуентност и външност.

## 5. ПРИМЕРИ ЗА РАБОТА НА АЛГОРИТЪМА

Следващите примери илюстрират работата на тази програма за извеждане типа на някои функции.

*Пример* 6. Да се намери типът на функцията идентитет $\mathrm{Id} = \lambda x.x$.

За целта чрез програмата TypeInfer ще оценим израза P( , **lam** x.x : ⟨t⟩), където променливата ⟨t⟩ е от тип ⟨**Tip**⟩ и означава търсения тип. Имаме

$$\begin{array}{c} \mathrm{P}(\ ,\mathbf{lam}\,\mathrm{x.x} : \langle \mathrm{t}\rangle) \\ \mathrm{T3} \downarrow (\langle \mathrm{t}\rangle = \langle \mathrm{t1}\rangle \to \langle \mathrm{t2}\rangle) \\ \mathrm{P}(\mathrm{x} : \langle \mathrm{t1}\rangle, \mathrm{x} : \langle \mathrm{t2}\rangle)\ \mathbf{if}\ (\langle \mathrm{t}\rangle = \langle \mathrm{t1}\rangle \to \langle \mathrm{t2}\rangle) \\ \mathrm{T1} \downarrow (\langle \mathrm{t2}\rangle = \langle \mathrm{t1}\rangle) \\ \mathbf{true}\ \mathbf{if}\ (\langle \mathrm{t}\rangle = \langle \mathrm{t1}\rangle \to \langle \mathrm{t1}\rangle) \end{array}$$

Следователно типът на функцията идентитет е ⟨t1⟩ → ⟨t1⟩, където t1 означава произволен тип.

Забелязваме, че в този случай горната програма напълно решава задачата и не е необходимо да се добавят допълнителни описания.

*Пример* 7. Да се намери типът на функцията f, така че f(x) = 2 ∗ x, ако типът на функцията ∗ е **nat** → **nat** → **nat**.

За целта ще оценим израза

$$\mathrm{P}(2 : \mathbf{nat}\ * : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}, \mathbf{lam}\ \mathrm{x}.(*\,2)\mathrm{x} : \langle \mathrm{t}\rangle),$$

където ⟨t⟩ е от тип ⟨**Tip**⟩ и означава търсения тип. Имаме

$$\begin{array}{c} \mathrm{P}(2 : \mathbf{nat}\ * : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat},\ \mathbf{lam}\ \mathrm{x}.(*2)\mathrm{x} : \langle \mathrm{t}\rangle) \\ \mathrm{T3} \downarrow (\langle \mathrm{t}\rangle = \langle \mathrm{t1}\rangle \to \langle \mathrm{t2}\rangle) \\ \mathrm{P}(2 : \mathbf{nat}\ * : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}\ \mathrm{x} : \langle \mathrm{t1}\rangle,\ (*2)\mathrm{x} : \langle \mathrm{t2}\rangle)\ \mathbf{if}\ (\langle \mathrm{t}\rangle = \langle \mathrm{t1}\rangle \to \langle \mathrm{t2}\rangle) \\ \mathrm{T4},\ \mathrm{T8},\ \mathrm{T4} \downarrow \\ \mathrm{P}(2 : \mathbf{nat}\ * : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}\ \mathrm{x} : \langle \mathrm{t1}\rangle,\ * : \mathbf{nat} \to \langle \mathrm{t1}\rangle \to \langle \mathrm{t2}\rangle)\ \mathbf{and} \\ \mathrm{P}(2 : \mathbf{nat}\ * : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}\ \mathrm{x} : \langle \mathrm{t1}\rangle,\ 2 : \mathbf{nat})\ \mathbf{and} \\ \mathrm{P}(2 : \mathbf{nat}\ * : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}\ \mathrm{x} : \langle \mathrm{t1}\rangle,\ \mathrm{x} : \langle \mathrm{t1}\rangle)\ \mathbf{if}\ (\langle \mathrm{t}\rangle = \langle \mathrm{t1}\rangle \to \langle \mathrm{t2}\rangle) \end{array}$$

$$T1 \downarrow (\langle t1\rangle = \texttt{nat},\ \langle t2\rangle = \texttt{nat})$$
$$\texttt{true and}$$
$$P(2 : \texttt{nat} * : \texttt{nat} \to \texttt{nat} \to \texttt{nat}\ \texttt{x} : \texttt{nat},\ 2 : \texttt{nat})\ \texttt{and}$$
$$P(2 : \texttt{nat} * : \texttt{nat} \to \texttt{nat} \to \texttt{nat}\ \texttt{x} : \texttt{nat},\ \texttt{x} : \texttt{nat})\ \texttt{if}\ (\langle t\rangle = \texttt{nat} \to \texttt{nat})$$
$$T9, T1, T9, T1 \downarrow$$
$$\texttt{true if}\ (\langle t\rangle = \texttt{nat} \to \texttt{nat})$$

Следователно при направените ограничения за типа на функцията $*$ типът на функцията f е $\texttt{nat} \to \texttt{nat}$.

Ако типът на функцията $*$ е $\texttt{nat} \to \texttt{real} \to \texttt{real}$, където real означава реалния тип, оценката на израза

$$P(2 : \texttt{nat} * : \texttt{nat} \to \texttt{real} \to \texttt{real}, \texttt{lam}\ \texttt{x}.(*2)\texttt{x} : \langle t\rangle)$$

преминава през същите стъпки и като резултат се получава $\texttt{true if}\ (\langle t\rangle = \texttt{real} \to \texttt{real})$. В този случай е необходимо в раздела на типовете дефиницията на типа $\langle V\rangle$ да се разшири с $\langle V\rangle :: = \texttt{real}$.

Ако типът на функцията $*$ е $\texttt{real} \to \texttt{nat} \to \texttt{real}$, оценката на израза $P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}, \texttt{lam}\ \texttt{x}.(*2)\texttt{x} : \langle t\rangle)$ преминава през следните стъпки:

$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real},\ \texttt{lam}\ \texttt{x}.(*2)\texttt{x} : \langle t\rangle)$$
$$T3 \downarrow (\langle t\rangle = \langle t1\rangle \to \langle t2\rangle)$$
$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}\ \texttt{x} : \langle t1\rangle,\ (*2)\texttt{x} : \langle t2\rangle)\ \texttt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t2\rangle)$$
$$T4 \downarrow$$
$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}\ \texttt{x} : \langle t1\rangle,\ (*2) : \langle t1\rangle \to \langle t2\rangle)\ \texttt{and}$$
$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}\ \texttt{x} : \langle t1\rangle,\ x : \langle t1\rangle)\ \texttt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t2\rangle)$$
$$T8, T4, T1 \downarrow$$
$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}\ \texttt{x} : \langle t1\rangle,\ * : \texttt{nat} \to \langle t1\rangle \to \langle t2\rangle)\ \texttt{and true and}$$
$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}\ \texttt{x} : \langle t1\rangle, \texttt{x} : \langle t1\rangle)\ \texttt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t2\rangle)$$
$$T10, T1, T10 \downarrow$$
$$P(2 : \texttt{nat} * : \texttt{real} \to \texttt{nat} \to \texttt{real}\ \texttt{x} : \langle t1\rangle,\ * : \texttt{nat} \to \langle t1\rangle \to \langle t2\rangle)$$
$$\texttt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t2\rangle)$$

Тъй като типовете за $*$ са $\texttt{real} \to \texttt{nat} \to \texttt{real}$ и $\texttt{nat} \to \langle t1\rangle \to \langle t2\rangle$ и не могат да се унифицират, последният резултат показва, че има грешка в типа на операцията $*$.

*Пример* 8. Да се намери типът на функцията g, дефинирана по следния начин: $g(f, x) = x + f(x)$, където типът на + е $\texttt{nat} \to \texttt{nat} \to \texttt{nat}$.

За целта ще оценим израза

$$P(+ : \texttt{nat} \to \texttt{nat} \to \texttt{nat},\ \texttt{lam}\ \texttt{f}.\texttt{lam}\ \texttt{x}.(+\texttt{x})(\texttt{f}\ \texttt{x}) : \langle t\rangle),$$

където $\langle t\rangle$ е променлива от тип $\langle \texttt{Tip}\rangle$ и означава търсения тип.

$$P(+ : \texttt{nat} \to \texttt{nat} \to \texttt{nat}\ (\texttt{f}\ \texttt{x}) : \langle t1\rangle,\ \texttt{lam}\ \texttt{f}.\texttt{lam}\ \texttt{x}.(+\texttt{x})(\texttt{f}\ \texttt{x}) : \langle t\rangle)$$
$$T3 \downarrow \langle t\rangle = \langle t2\rangle \to \langle t3\rangle$$
$$P(+ : \texttt{nat} \to \texttt{nat} \to \texttt{nat}\ (\texttt{f}\ \texttt{x}) : \langle t1\rangle\ \texttt{f} : \langle t2\rangle,\ \texttt{lam}\ \texttt{x}.(+\texttt{x})(\texttt{f}\ \texttt{x}) : \langle t3\rangle)\ \texttt{if}\ (\langle t\rangle = \langle t2\rangle \to \langle t3\rangle)$$
$$T3 \downarrow \langle t3\rangle = \langle t4\rangle \to \langle t5\rangle$$

$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \langle t1\rangle\ \mathtt{f} : \langle t2\rangle.\mathtt{x} : \langle t4\rangle,\ (+\mathtt{x})(\mathtt{f}\ \mathtt{x}) : \langle t5\rangle)\ \mathtt{if}\ (\langle t\rangle = \langle t2\rangle \to \langle t4\rangle \to \langle t5\rangle)$$

T4, T8, T4 ↓

$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \langle t1\rangle \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \langle t4\rangle,\ + : \langle t4\rangle \to \langle t1\rangle \to \langle t5\rangle)\ \mathtt{and}$$
$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \langle t1\rangle\ \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \langle t4\rangle,\ \mathtt{x} : \langle t4\rangle)\ \mathtt{and}$$
$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \langle t1\rangle \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \langle t4\rangle,\ (\mathtt{f}\ \mathtt{x}) : \langle t1\rangle)\ \mathtt{if}\ (\langle t\rangle = \langle t2\rangle \to \langle t4\rangle \to \langle t5\rangle)$$

T1 ↓ ($\langle t1\rangle = \mathtt{nat}$, $\langle t4\rangle = \mathtt{nat}$, $\langle t5\rangle = \mathtt{nat}$)

$$\mathtt{true}\ \mathtt{and}\ P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \mathtt{nat}\ \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \mathtt{nat},\ \mathtt{x} : \mathtt{nat})\ \mathtt{and}$$
$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \mathtt{nat}\ \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \mathtt{nat},\ (\mathtt{f}\ \mathtt{x}) : \mathtt{nat})\ \mathtt{if}\ (\langle t\rangle = \langle t2\rangle \to \mathtt{nat} \to \mathtt{nat})$$

T9, T1, T9, T8, T4 ↓

$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \mathtt{nat}\ \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \mathtt{nat},\ \mathtt{f} : \mathtt{nat} \to \mathtt{nat})\ \mathtt{and}$$
$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \mathtt{nat}\ \mathtt{f} : \langle t2\rangle\ \mathtt{x} : \mathtt{nat},\ \mathtt{x} : \mathtt{nat})\ \mathtt{if}\ (\langle t\rangle = \langle t2\rangle \to \mathtt{nat} \to \mathtt{nat})$$

T1 ↓ $\langle t2\rangle = \mathtt{nat} \to \mathtt{nat}$

$$\mathtt{true}\ \mathtt{and}$$
$$P(+ : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ (\mathtt{f}\ \mathtt{x}) : \mathtt{nat}\ \mathtt{f} : \mathtt{nat} \to \mathtt{nat}\ \mathtt{x} : \mathtt{nat},\ \mathtt{x} : \mathtt{nat})$$
$$\mathtt{if}\ (\langle t\rangle = (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat} \to \mathtt{nat})$$

Получената оценка показва, че типът на g е $(\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat} \to \mathtt{nat}$.

*Пример* 9. Да се намери типът на функцията `map`.

За целта ще оценим израза

$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle](\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle,$$
$$\mathtt{fix\ map.lam\ f.lam\ l.if\ (null\ l)\ then\ [\,]\ else\ cons\ (f\ (hd\ l),\ (map\ f)(tl\ l))} : \langle t\rangle)$$

T5 ↓

$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle$$
$$\mathtt{map} : \langle t\rangle, \mathtt{lam\ f.lam\ l.if\ (null\ l)\ then\ [\,]\ else\ cons\ (f\ (hd\ l), (map\ f)(tl\ l))} : \langle t\rangle)$$

T3 ↓ $\langle t\rangle = \langle t1\rangle \to \langle t2\rangle$

$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]$$
$$(\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle\ \mathtt{map} : \langle t1\rangle \to \langle t2\rangle\ \mathtt{f} : \langle t1\rangle,$$
$$\mathtt{lam\ l.\ if\ (null\ l)\ then\ [\,]\ else\ cons\ (f\ (hd\ l), (map\ f)(tl\ l))} : \langle t2\rangle)$$
$$\mathtt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t2\rangle)$$

T3 ↓ $\langle t2\rangle = \langle t3\rangle \to \langle t4\rangle$

$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]$$
$$(\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle\ \mathtt{map} : \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle\ \mathtt{f} : \langle t1\rangle,$$
$$\mathtt{l} : \langle t3\rangle,\ \mathtt{if\ (null\ l)\ then\ [\,]\ else\ cons\ (f\ (hd\ l), (map\ f)(tl\ l))} : \langle t4\rangle)$$
$$\mathtt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle)$$

T7, T8, T4 ↓

$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle$$
$$(\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle\ \mathtt{map} : \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle\ \mathtt{f} : \langle t1\rangle\ \mathtt{l} : \langle t3\rangle,\ \mathtt{null} : \langle t3\rangle \to \mathtt{bool})$$
$$\mathtt{and}$$
$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle$$
$$\mathtt{map} : \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle\ \mathtt{f} : \langle t1\rangle\ \mathtt{l} : \langle t3\rangle,\ \mathtt{l} : \langle t3\rangle)$$
$$\mathtt{and}$$
$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle$$
$$\mathtt{map} : \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle\ \mathtt{f} : \langle t1\rangle\ \mathtt{l} : \langle t3\rangle,\ [\,] : \langle t4\rangle)$$
$$\mathtt{and}$$
$$P(\mathtt{hd} : [\langle z\rangle] \to \langle z\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ \mathtt{l}) : \langle r1\rangle\ (\mathtt{tl}\ \mathtt{l}) : \langle r2\rangle$$
$$\mathtt{map} : \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle\ \mathtt{f} : \langle t1\rangle\ \mathtt{l} : \langle t3\rangle,\ \mathtt{cons\ (f\ (hd\ l),\ (map\ f)(tl\ l))} : \langle t4\rangle)$$
$$\mathtt{if}\ (\langle t\rangle = \langle t1\rangle \to \langle t3\rangle \to \langle t4\rangle)$$

T1, T9, T1 ↓

true and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → ⟨t4⟩ f : ⟨t1⟩ l : [⟨m⟩], [] : ⟨t4⟩)

and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩]
(hd l) : ⟨r1⟩ (tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → ⟨t4⟩ f : ⟨t1⟩ l : [⟨m⟩],
cons (f (hd l), (map f)(tl l)) : ⟨t4⟩)
if (⟨t⟩ = ⟨t1⟩ → [⟨m⟩] → ⟨t4⟩)

T9, T1 ↓ ⟨t4⟩ = [⟨n⟩]

true and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩]
(hd l) : ⟨r1⟩ (tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → ⟨n⟩ f : ⟨t1⟩.l : [⟨m⟩],
cons (f (hd l), (map f)(tl l)) : [⟨n⟩])
if (⟨t⟩ = ⟨t1⟩ → [⟨m⟩] → [⟨n⟩])

T9 ↓

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩]
(hd l) : ⟨r1⟩ (tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → ⟨n⟩ f : ⟨t1⟩ l : [⟨m⟩],
cons(f (hd l), (map f)(tl l)) : [⟨n⟩])
if (⟨t⟩ = ⟨t1⟩ → [⟨m⟩] → [⟨n⟩])

CONS-правило, T4 ↓

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → ⟨n⟩ f : ⟨t1⟩ l : [⟨m⟩], f : ⟨r1⟩ → ⟨n⟩)

and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → ⟨n⟩ f : ⟨t1⟩ l : [⟨m⟩], (hd l) : ⟨r1⟩)

and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : ⟨t1⟩ → [⟨m⟩] → [⟨n⟩] f : ⟨t1⟩ l : [⟨m⟩], (map f)(tl l) : [⟨n⟩])
if (⟨t⟩ = ⟨t1⟩ → [⟨m⟩] → [⟨n⟩])

T1 ↓ ⟨t1⟩ = ⟨r1⟩ → ⟨n⟩

true and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : (⟨r1⟩ → ⟨n⟩) → [⟨m⟩] → ⟨n⟩ f : ⟨r1⟩ → ⟨n⟩ l : [⟨m⟩], (hd l) : ⟨r1⟩)

and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : (⟨r1⟩ → ⟨n⟩) → [⟨m⟩] → [⟨n⟩] f : ⟨r1⟩ → ⟨n⟩ l : [⟨m⟩], (map f)(tl l) : [⟨n⟩])
if (⟨t⟩ = ⟨r1⟩ → ⟨n⟩ → [⟨m⟩] → [⟨n⟩])

T9, T8, T4 ↓

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : (⟨r1⟩ → ⟨n⟩) → [⟨m⟩] → [⟨n⟩] f : ⟨r1⟩ → ⟨n⟩ l : [⟨m⟩], hd : [⟨m⟩] → ⟨r1⟩)

and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩.
(tl l) : ⟨r2⟩ map : (⟨r1⟩ → ⟨n⟩) → [⟨m⟩] → ⟨t4⟩ f : ⟨r1⟩ → ⟨n⟩ l : [⟨m⟩], l : [⟨m⟩])

and

P(hd : [⟨z⟩] → ⟨z⟩ tl : [⟨y⟩] → [⟨y⟩] null : [⟨m⟩] → bool [] : [⟨n⟩] (hd l) : ⟨r1⟩
(tl l) : ⟨r2⟩ map : (⟨r1⟩ → ⟨n⟩) → [⟨m⟩] → [⟨n⟩] f : ⟨r1⟩ → ⟨n⟩) l : [⟨m⟩], (map f)(tl l) : [⟨n⟩])
if (⟨t⟩ = (⟨r1⟩ → ⟨n⟩) → [⟨m⟩] → [⟨n⟩])

T1, T9 ↓ (⟨r1⟩ = ⟨m⟩, ⟨z⟩ = ⟨m⟩)

$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd\ l}) : \langle r1\rangle$$
$$(\mathtt{tl\ l}) : \langle r2\rangle\ \mathtt{map} : (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle r1\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle], (\mathtt{map\ f})(\mathtt{tl\ l}) : [\langle n\rangle])$$
$$\mathtt{if}\ (\langle t\rangle = (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle];\ \langle z\rangle = \langle m\rangle)$$
$$\text{T4, T8, T4} \downarrow$$
$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ l) : \langle m\rangle$$
$$(\mathtt{tl\ l}) : \langle r2\rangle\ \mathtt{map} : (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle m\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle],$$
$$\mathtt{map} : \langle m\rangle \to \langle n\rangle \to \langle r2\rangle \to [\langle n\rangle])$$
$$\textbf{and}$$
$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd}\ l) : \langle m\rangle$$
$$(\mathtt{tl\ l}) : \langle r2\rangle\ \mathtt{map} : (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle m\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle],\ \mathtt{f} : \langle m\rangle \to \langle n\rangle)$$
$$\textbf{and}$$
$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd\ l}) : \langle m\rangle$$
$$(\mathtt{tl\ l}) : \langle r2\rangle\ \mathtt{map} : \langle m\rangle \to \langle n\rangle \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle r1\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle],\ (\mathtt{tl\ l}) : \langle r2\rangle)$$
$$\mathtt{if}\ (\langle t\rangle = (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle];\ \langle z\rangle = \langle m\rangle)$$
$$\text{T1, T9} \downarrow \langle r2\rangle = [\langle m\rangle]$$
$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd\ l}) : \langle m\rangle$$
$$(\mathtt{tl\ l}) : [\langle m\rangle]\ \mathtt{map} : (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle m\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle],\ (\mathtt{tl\ l}) : [\langle m\rangle])$$
$$\mathtt{if}\ (\langle t\rangle = (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle];\ \langle z\rangle = \langle m\rangle)$$
$$\text{T8, T4} \downarrow$$
$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd\ l}) : \langle m\rangle$$
$$(\mathtt{tl\ l}) : [\langle m\rangle]\ \mathtt{map} : (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle m\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle],\ \mathtt{tl} : [\langle m\rangle] \to [\langle m\rangle])$$
$$\textbf{and}$$
$$\mathtt{P(hd} : [\langle m\rangle] \to \langle m\rangle\ \mathtt{tl} : [\langle y\rangle] \to [\langle y\rangle]\ \mathtt{null} : [\langle m\rangle] \to \mathtt{bool}\ [\,] : [\langle n\rangle]\ (\mathtt{hd\ l}) : \langle m\rangle$$
$$(\mathtt{tl\ l}) : [\langle m\rangle]\ \mathtt{map} : \langle m\rangle \to \langle n\rangle \to [\langle m\rangle] \to [\langle n\rangle]\ \mathtt{f} : \langle m\rangle \to \langle n\rangle)\ \mathtt{l} : [\langle m\rangle],\ \mathtt{l} : [\langle m\rangle])$$
$$\mathtt{if}\ (\langle t\rangle = (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle];\ \langle z\rangle = \langle m\rangle)$$
$$\text{T1, T9} \downarrow \langle y\rangle = \langle m\rangle$$
$$\mathtt{true\ if}\ (\langle t\rangle = (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle];\ \langle z\rangle = \langle m\rangle;\ \langle y\rangle = \langle m\rangle)$$

Резултатът от оценката показва, че типът на функцията **map** е $(\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]$, където $\langle m\rangle$ и $\langle n\rangle$ са произволни типове. За реализирането на тази оценка е необходимо да се допълни обобщеният трансформатор TypeInfer с още една трансформация, която наричаме cons-правило:

$$\{\text{T13}\}\quad \mathtt{P}(\langle a\rangle, \mathtt{cons}(\langle e1\rangle, \langle e2\rangle) : [\langle t\rangle]) \Longrightarrow \mathtt{P}(\langle a\rangle, \langle e1\rangle : \langle t\rangle)\ \textbf{and}\ \mathtt{P}(\langle a\rangle, \langle e2\rangle : [\langle t\rangle])$$

В раздела на променливите трябва да се добави декларацията на променливите $\langle z\rangle$, $\langle y\rangle$, $\langle m\rangle, \langle n\rangle$, а в раздела на типовете — дефиницията на cons-израз.

*Пример 10.* *Да се намери типът на израза*

$$\mathtt{let\ square} = \lambda x.(*x)x\ \mathtt{in}\ (\mathtt{map\ square})l.$$

Знаем, че типът на функцията **map** е $(\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]$, където $\langle m\rangle$ и $\langle n\rangle$ са произволни типове. За да намерим типа на горния израз, ще оценим израза $\mathtt{P(map} : (\langle m\rangle \to \langle n\rangle) \to [\langle m\rangle] \to [\langle n\rangle]\ * : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}\ \ l : [\langle s\rangle]\ \mathtt{lam\ x.(*x)x} : \langle R\rangle,\ \mathtt{let\ square} = \mathtt{lam\ x.(*x)x\ in\ (map\ square)}\ l : \langle t\rangle)$ чрез обобщения трансформатор TypeInfer:

$$\mathrm{P}(\texttt{map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle] * : \texttt{nat} \to \texttt{nat} \to \texttt{nat}\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{R}\rangle,$$
$$\texttt{let square} = \texttt{lam x.}(*\texttt{x})\texttt{x in (map square)}\ \mathrm{l} : \langle \mathrm{t}\rangle)$$
$$\text{T6, T3} \downarrow \langle \mathrm{R}\rangle = \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle$$
$$\mathrm{x} : \langle \mathrm{r1}\rangle,\ (*\texttt{x})\mathrm{x} : \langle \mathrm{r2}\rangle)\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle.$$
$$\texttt{square} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle,\ \texttt{(map square)}\ \mathrm{l} : \langle \mathrm{t}\rangle)\ \texttt{if}\ (\langle \mathrm{R}\rangle = \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle)$$
$$\text{T4, T8, T4} \downarrow$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle$$
$$\mathrm{x} : \langle \mathrm{r1}\rangle,\ * : \langle \mathrm{r1}\rangle \to \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle)\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle$$
$$\mathrm{x} : \langle \mathrm{r1}\rangle,\ \mathrm{x} : \langle \mathrm{r1}\rangle)\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle$$
$$\mathrm{x} : \langle \mathrm{r1}\rangle,\ \mathrm{x} : \langle \mathrm{r1}\rangle)\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle$$
$$\texttt{square} : \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle,\ \texttt{(map square)}\ \mathrm{l} : \langle \mathrm{t}\rangle)\ \texttt{if}\ (\langle \mathrm{R}\rangle = \langle \mathrm{r1}\rangle \to \langle \mathrm{r2}\rangle)$$
$$\text{T1, T9} \downarrow (\langle \mathrm{r1}\rangle = \texttt{nat};\ \langle \mathrm{r2}\rangle = \texttt{nat})$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \texttt{nat} \to \texttt{nat}$$
$$\texttt{square} : \texttt{nat} \to \texttt{nat},\ \texttt{(map square)}\ \mathrm{l} : \langle \mathrm{t}\rangle)$$
$$\texttt{if}\ (\langle \mathrm{R}\rangle = \texttt{nat} \to \texttt{nat})$$
$$\text{T4, T8, T4} \downarrow$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \texttt{nat} \to \texttt{nat}$$
$$\texttt{square} : \texttt{nat} \to \texttt{nat},\ \texttt{map} : (\texttt{nat} \to \texttt{nat}) \to [\langle \mathrm{s}\rangle] \to \langle \mathrm{t}\rangle)\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \texttt{nat} \to \texttt{nat}$$
$$\texttt{square} : \texttt{nat} \to \texttt{nat},\ \texttt{square} : \texttt{nat} \to \texttt{nat})\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\langle \mathrm{m}\rangle \to \langle \mathrm{n}\rangle) \to [\langle \mathrm{m}\rangle] \to [\langle \mathrm{n}\rangle]\ \mathrm{l} : [\langle \mathrm{s}\rangle]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \texttt{nat} \to \texttt{nat}$$
$$\texttt{square} : \texttt{nat} \to \texttt{nat},\ \mathrm{l} : [\langle \mathrm{s}\rangle])\ \texttt{if}\ (\langle \mathrm{R}\rangle = \texttt{nat} \to \texttt{nat})$$
$$\text{T1} \downarrow (\langle \mathrm{m}\rangle = \texttt{nat},\ \langle \mathrm{n}\rangle = \texttt{nat},\ \langle \mathrm{s}\rangle = \texttt{nat},\ \langle \mathrm{t}\rangle = [\texttt{nat}])$$
$$\texttt{true and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : (\texttt{nat} \to \texttt{nat}) \to [\texttt{nat}] \to [\texttt{nat}]\ \mathrm{l} : \texttt{nat}\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \texttt{nat} \to \texttt{nat}$$
$$\texttt{square} : \texttt{nat} \to \texttt{nat},\ \texttt{square} : \texttt{nat} \to \texttt{nat})\ \texttt{and}$$
$$\mathrm{P}(* : \texttt{nat} \to \texttt{nat} \to \texttt{nat map} : \texttt{nat} \to \texttt{nat} \to [\texttt{nat}] \to [\texttt{nat}]\ \mathrm{l} : [\texttt{nat}]\ \texttt{lam x.}(*\texttt{x})\texttt{x} : \texttt{nat} \to \texttt{nat}$$
$$\texttt{square} : \texttt{nat} \to \texttt{nat},\ \mathrm{l} : [\texttt{nat}])\ \texttt{if}\ (\langle \mathrm{R}\rangle = \texttt{nat} \to \texttt{nat}),\ \langle \mathrm{s}\rangle = \texttt{nat}, \langle \mathrm{t}\rangle = [\texttt{nat}])$$
$$\text{T9, T1, T9} \downarrow$$
$$\texttt{true, if}\ (\langle \mathrm{R}\rangle = \texttt{nat} \to \texttt{nat},\ \langle \mathrm{t}\rangle = [\texttt{nat}],\ \langle \mathrm{s}\rangle = \texttt{nat})$$

Получената оценка показва, че **square** е от тип **nat** → **nat**, а типът на дадения израз е [**nat**]. Освен това е необходимо типът на дадения списък l да е [**nat**].

## 6. ЗАКЛЮЧЕНИЕ

Алгоритъмът за извод на тип на Milner извежда типа на даден израз, след като изведе типовете на всички негови подизрази — редекси. Ако представим $\lambda$-израза във вид на дърво, започвайки от листата, процесът на извод се насочва към корена на дървото или протича отвътре навън. В предложената от нас реализация процесът на извод на типа, започвайки

от корена, се движи към листата на дървото, като постепенно се уточнява типът на основния терм — или отвън навътре.

За програмите на езика W можем да доказваме определени свойства. Ето защо езикът W представлява удобен инструмент за изследване свойствата на различни системи за извод на тип.

## ЛИТЕРАТУРА


1. Barendregt, H. The Lambda calculus: Its syntax and semantics. North Holland, Amsterdam, 1984.
2. Cardelli, L., P. Wegner. On understanding types, data abstraction and polymorphism. *ACM Computing Surveys*, **17**, 4, Dec. 1985, 471–522.
3. Damas, L., R. Milner. Principal type schemes for functional programs. In: Proceedings of the 9th ACM Symposium on Principles of Programming Languages, 1982, 207–212.
4. Hindley, R., J. Seldin. Introduction to combinators and $\lambda$-calculus. Vol. 1, London Mathematical Society Students Texts, Cambridge University Press, 1986.
5. Hudak, P., S. P. Jones, P. Wadler. Report on the Programming Language Haskell: Version 1.1. Technical Report, *Yale University and Glasgow University*, August 1991.
6. Liskov, B., J. Guttag. Abstraction and Specification in Program Development. The Massachusets Institute of Technology, 1986. Също превод на руски език: Использование абстракций и спецификаций при разработке программ, М., Мир, 1989.
7. Milner, R. A theory of type polymorphism in programming. *J. of Computer and System Sciences*, **17**, 1978, 348–375.
8. Milner, R., M. Tofte, R. Harper. The Definition of Standard ML. MIT Press, 1990.
9. Mitchell, J. Type Systems for Programming Languages. In: Handbook of Theoretical Computer Science, ed. J. van Leeuwen, North-Holland, Amsterdam, 1990.
10. Mycroft, A. Polymorphic type schemes and recursive definitions. In: Proceedings of the 6th International Conference on Programming, *LNCS*, **167**, 1984.
11. Radensky, A., M. Todorova. An approach to programming by means of equations: transformation programs and an interpreter for such programs. In: Конференция КНВВТ по автоматизации информационных процессов на персональных ЭВМ, Будапест, 1986, 167–181.
12. Robinson, J. A. A machine-oriented logic based on resolution principle. *J. of ACM*, **12**, 1965, 23–41.
13. Todorova, M. W: functional programming language based on unification. *Informatika*, **21**, 3, 1987, 143–151.
14. Todorova, M. Determination classes of reduction sequences in generalized subtree replacement systems. *Ann. Sof. Univ.*, FMI, **79**, 1985, 127–133.
15. Todorova, M. Generalized subtree replacement systems. PhD. Thesis, University of Sofia, Faculty of Mathematics and Informatics, 1987.
16. Todorova, M. The W language and its applications to theorem proving. *SERDICA*, Bulgaricae Mathematicae Publicationes, **19**, 1993, 36–44.
17. Turner, D. A. Miranda: A non-strict functional language with polymorphic types. In: IFIP International Conference on Functional Programming and Computer Architecture. *LNCS*, **201**, Springer-Verlag, N. Y., 1985, 54–64.
18. Turner, D. A. An Overview of Miranda. In: D. A. Turner, ed., Research Topics in Functional Programming, Addison Wesley, 1990.


*Постъпила 15.02.1994*